CLAIMS

What is claimed is:

1. A method for processing and organizing binary data, the method comprising:

splitting an extended opcode from an instruction, and

storing an index code in an opcode list as reference to an instruction entry in an instruction format dictionary representing the extended opcode, wherein the index code is stored in the order that the instruction is provided in an instruction stream.

2. The method of claim 1, wherein the instruction further has at least one parameter and the method further includes:

splitting a parameter of a particular type from the instruction, and

storing the parameter in a parameter list for the particular type.

3. The method of claim 2, further including repeating splitting of the parameter and the storing of the parameter, for another parameter.

4. The method of claim 2, wherein the particular type is an immediate value, an offset, or a register index.

5. The method of claim 1, wherein the instruction entry includes at least one instruction prefix byte and opcode.

6. The method of claim 1, further including adding the extended opcode as an instruction entry in the instruction format dictionary.

7. The method of claim 1, wherein the instruction has at least one parameter and the method further includes:

considering the parameter to identify a similar use property of the parameter;

splitting the parameter; and

storing the parameter in a sub-list for the similar use property.

8. The method of claim 7, further including updating the instruction format dictionary with the sub-list.

9. The method of claim 7, wherein the considering is by applying rules specific to the binary data.

10. The method of claim 9, wherein the rules are formulated by statistical analysis of the parameters of a particular type in the entries of the instruction format dictionary.

11. The method of claim 7, wherein the similar use property is any one of a group including size, sign, number of bits required to store the parameter, and module 2^n class.

12. The method of claim 1, wherein the instruction has at least one parameter and the method further includes:

considering the parameter to identify a random use property of the parameter;

splitting the parameter having a random use property; and

storing the parameter in a dissimilar sub-list for the random use property.

13. The method of claim 12, further including updating the instruction format dictionary with the dissimilar sub-list.

14. A crushing unit for processing and organizing binary data, the unit comprising:

a decomposition module to split an extended opcode from an instruction;

an instruction format dictionary to store the extended opcode and an index code as an entry; and

an opcode list having the index code in the order that the instruction is provided in an instruction stream.

15. The unit of claim 14, wherein the decomposition module is further to split at least one parameter of a particular type from the instruction and the crushing unit further includes at least one parameter list to store the parameter.

16. The unit of claim 15, wherein the type is any one of a group including an immediate value, an offset, and a register index.

17. The unit of claim 15, wherein the entry includes at least one instruction prefix byte and opcode.

18. The unit of claim 14, further including an analysis module to consider a parameter in the instruction to identify a subtype of a similar use property of the parameter.

19. The unit of claim 18, wherein the analysis module is further to split the parameter and the unit further including a sub-list to store the parameter with the subtype.

20. The unit of claim 19, wherein the instruction format dictionary is further to store the parameter subtype in the entry having the extended opcode associated with the parameter.

21. The unit of claim 18, wherein the considering is by applying rules specific to the binary data.

22. The unit of claim 21, wherein the rules are formulated by statistical analysis of the parameters of a particular type in the entries of the instruction format dictionary.

23. The unit of claim 18, wherein the similar use property is any one of a group including size, sign, number of bits required to store the parameter, and module 2^n class.

24. The unit of claim 14, further including a dissimulation module to consider a parameter in the instruction to identify a random use property of the parameter.

25. The unit of claim 24, wherein the dissimulation module further splits the parameter having the random use property and the crushing unit further including a dissimilar sub-list to store the parameter having the random use property.

26. The unit of claim 25, wherein the instruction format dictionary further includes the parameter having the random use property in the entry for the extended opcode associated with the parameter.

27. A computer readable medium having stored therein a plurality of sequences of executable instructions, which, when executed by a processing system for processing and organizing binary data, cause the system to:

split an extended opcode from an instruction, and

store an index code in an opcode list as reference to an instruction entry in an instruction format dictionary representing the extended opcode, wherein the index code is stored in the order that the instruction is provided in an instruction stream.

28. The computer readable medium of claim 27, further including additional sequences of executable instructions, which, when executed by the system, cause the system to split a parameter of a particular type from the instruction, and store the parameter in a parameter list for the particular type.

29. The computer readable medium of claim 27, further including additional sequences of executable instructions, which, when executed by the system, cause the system to identify a similar use property of a parameter in the instruction, split the parameter and store the parameter in a sub-list for the similar use property.

30. The computer readable medium of claim 29, further including additional sequences of executable instructions, which, when executed by the system, cause the system to update the instruction format dictionary with the sub-list.

31. A method for processing and organizing binary data in an executable file, the method comprising:

replacing the symbol name corresponding to the dynamic link symbol in the binary data with an abbreviation that is smaller than the symbol name, and

writing the symbol name and abbreviation as an entry in a symbol dictionary.

32. The method of claim 31, further including publishing an exception symbol name that is directly accessed by an executable during execution.

33. The method of claim 31, wherein the symbol dictionary is extendable to later include a new dynamic link symbol as another entry in the symbol dictionary.

34. The method of claim 31, further including eliminating a symbol hash table from the executable file.

35. The method of claim 31, further including:

identifying an unused symbol to export by the executable file and not to import by another executable file;

writing an unused symbol name associated with the unused symbol as an entry in the symbol dictionary; and

eliminating the unused symbol and unused symbol name.

36. The method of claim 35, further including marking the unused symbol as unused in the symbol dictionary.

37. The method of claim 35, further including generating a previously eliminated unused symbol by accessing the unused symbol name from the unused symbol dictionary.

38. A crushing unit for processing and organizing binary data in an executable file, the unit comprising:

a renaming module to replace a symbol name corresponding to a dynamic link symbol in the binary data with an abbreviation that is smaller than the symbol name, and

a symbol dictionary having an entry including the symbol name and abbreviation.

39. The unit of claim 38, wherein the renaming module is further to publish an exception symbol name that is directly accessed by an executable during execution.

40. The unit of claim 38, wherein the symbol dictionary is extendable to later include a new dynamic link symbol as another entry in the symbol dictionary.

41. The unit of claim 38, wherein the elimination module is further to remove a symbol hash table from the executable file.

42. The unit of claim 38, further including an elimination module to identify and to remove an unused symbol for exporting by the executable file and not importing by another executable file and the symbol dictionary further includes an unused symbol name associated with the unused symbol as an entry.

43. The unit of claim 42, wherein the symbol dictionary further includes a notation of the unused status of the unused symbol.

44. The unit of claim 42, wherein the unused symbol name in the symbol dictionary is further to be accessible to generate a previously eliminated unused symbol.

45. A computer readable medium having stored therein a plurality of sequences of executable instructions, which, when executed by a processing system for processing and organizing binary data, cause the system to:

replace the symbol name corresponding to the dynamic link symbol in the binary data with an abbreviation that is smaller than the symbol name, and

write the symbol name and abbreviation as an entry in a symbol dictionary.

46. The computer readable medium of claim 45, further including additional sequences of executable instructions, which, when executed by the system, cause the system to publish an exception symbol name that is directly accessed by an executable during execution.

47. The computer readable medium of claim 45, further including additional sequences of executable instructions, which, when executed by the system, cause the system to:

identify an unused symbol to export by the executable file and not to import by another executable file;

write an unused symbol name associated with the unused symbol as an entry in the symbol dictionary; and

eliminate the unused symbol and unused symbol name.

48. The computer readable medium of claim 47, further including additional sequences of executable instructions, which, when executed by the system, cause the system to mark the unused symbol as unused in the symbol dictionary.